Developing Reliable ADAS and Autonomous Driving Functions in MATLAB and Simulink

MathWorks
AUTOMOTIVE
CONFERENCE 2016

Marco Roggero
Senior Application Engineer
marco.roggero@mathworks.de
The MathWorks environment helps ADAS engineers …

- Gain insight by replaying and visualizing logged and synthetic vehicle data
- Efficiently model components and develop processing algorithms
- Speed up prototyping of algorithms by generating code
Agenda

Workflow for ADAS algorithm development
  - Forward Collision Warning example
  - Sensor Fusion techniques

Sensor components modelling and simulation
  - Radars
  - Lidars
  - Cameras

Ground Truth Labeling
Integration in Model Based Design workflow
How do I know my ADAS algorithm is robust enough?

- Worst-case scenarios
- Scenarios identified from real world test drive data
- OEM specific test scenarios
- Fail Operation test scenarios
Example workflow for ADAS algorithm development

1. Create new traffic scenario or refine sensor model
2. Drive and collect more vehicle data
3. Refine algorithm
4. Expected behavior?
   - yes: Generate code
   - no: Integrate with embedded environment

Synthetic data → ADAS algorithm → Expected behavior? → yes

Logged vehicle data → ADAS algorithm → Expected behavior? → no

Create new traffic scenario or refine sensor model → ADAS algorithm

C Code
Gain insight by *replaying and visualizing logged vehicle data*

- Synthetic data
  - ADAS algorithm
    - Expected behavior?
      - yes
      - no
        - Refine algorithm
          - Drive and collect more vehicle data
            - Create new traffic scenario or refine sensor model
  - Generate code
    - C Code
      - Integrate with embedded environment
Test vehicle equipped with sensors

**Velodyne LiDAR HDL-32E**
- Horizontal FoV: 360°
- Vertical FoV: +10°..-30°
- Range: 80..100m
- 100 Mbps Ethernet

**Mobileye 560**
- FoV: 38°x150m
- CAN interface

**Point Grey Blackfly**
- Stand “ice cube” vision camera
- 800x600, 27FPS
- GigE interface

**XSENS MTI-G-700**
- Stable and sensitive
- MEMS-based AHRS
- USB interface

**Delphi ESR**
- 76GHz electronically scanning radar
- Dual FoVs, 90°x60m, 20°x174m
- CAN interface
Example test scenarios in public road

01_city_c2s_fcw

02_city_stopngo

03_local_streetParking

04_highway_cornering

05_highway_lanechange

06_highway_cutin
Sensor fusion algorithm for FCW

Data Pre-processing

Path Estimation

Sensor Fusion & Tracking

Threat Assessment

Radar Object

Vision Object

Vision LD

Vehicle CAN

FCW

MIO: Most-Important Object
Sensor fusion algorithm for FCW

Data Pre-processing
- Calculate Ground Speed
- Object classification

Path Estimation
- Zoning

Sensor Fusion & Tracking
- Sensor Fusion
- Kalman Filter

Path Estimation

Threat Assessment
- Maneuver Analysis
- Risk Assessment
- Find MIO

FCW

MIO: Most-Important Object
Sensor fusion algorithm for FCW

Data Pre-processing
- Calculate Ground Speed
- Object classification

Path Estimation
- Zoning

Sensor Fusion & Tracking
- Sensor Fusion
- Kalman Filter

Threat Assessment
- Maneuver Analysis
- Risk Assessment
- Find MIO
- Zoning

MIO: Most-Important Object

- Radar Object
- Vision Object
- Vision LD
- Vehicle CAN
Sensor fusion

Birds-Eye View object notations
- Radar object (stationary)
- Radar object (moving)
- Vision object
- Fused object (safe zone)
- Fused object (warn zone)
- Fused object (FCW zone)
- Fused object (most important object)
Sensor Fusion Made Easy by MATLAB CVST

Sensor fusion involves the combination of data from multiple sensors to improve the accuracy and reliability of the information. In the context of object detection and tracking, this process can be visualized as follows:

- **Vision Objects** and **Radar Objects** are the inputs to the sensor fusion process.
- A **costMatrix** is used to calculate the costs associated with each possible assignment of a vision object to a radar object.
- The `assignDetectionsToTracks` function is used to find the optimal assignments based on the costMatrix.

The algorithm can be represented as:

```
[assignments, unassignedVisions, unassignedRadars] = ...
assignDetectionsToTracks(costMatrix, param.costOfNonAssignment);
```

The assignments result in a list of **Fused Object List**.

Pairs of visions and associated radars are determined by the assignments made.

Fusion of the individual detection and tracking results into a single, coherent set of objects is achieved through the following formula:

\[
\text{Fusion} = f(V_1) + f(R_1) + f(V_2) + f(R_2) + \ldots + f(V_n) + f(R_m)
\]
Object Tracking by Kalman Filter

- Data Pre-processing
  - Calculate Ground Speed
  - Object classification
- Path Estimation
- Sensor Fusion & Tracking
- Threat Assessment
  - Maneuver Analysis
  - Risk Assessment
  - Find MIO

Radar Object
Vision Object
Vision LD
Vehicle CAN

MIO: Most-Important Object
What is the Kalman Filter?

- It is an iterative mathematical process that uses a set of equations and consecutive data inputs to quickly estimate the true value, position, velocity, etc. of the object being measured, when the measured values contain random noise.
Challenges in object tracking

State Matrix

\[
\begin{bmatrix}
  x \\
  \dot{x} \\
  y \\
  \dot{y}
\end{bmatrix}
\]

predict

Motion Model

correct

Measurement
Kalman Filter

**Initial state & covariance**

\[ \hat{x}_0, P_0 \]

**Previous state & covariance**

\[ \hat{x}_{k-1}, P_{k-1} \]

**Time Update (“Predict”)**

1. Predict state based on physical model and previous state
   \[ \hat{x}_k^\sim = A\hat{x}_{k-1} + Bu_k + w_k \]
2. Predict error covariance matrix
   \[ P_k^- = AP_{k-1}A^T + Q \]

**Measurement Update (“Correct”)**

1. Compute Kalman gain
   \[ K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \]
2. Update estimate state with measurement
   \[ \hat{x}_k = \hat{x}_k^- + K_k (z_k - H\hat{x}_k^-) \]
3. Update the error covariance matrix
   \[ P_k = (I - K_k H)P_k^- \]

**Symbols**

- **\( u \)**: Control variable matrix
- **\( w \)**: Process (state) noise
- **\( P_k^- \)**: Process (state) covariance matrix
- **\( e_k^- \)**: Process (state) covariance matrix (estimation error)
- **\( Q = E[ww^T] \)**: Process noise covariance matrix
- **\( A \)**: State matrix relates the state at the previous, \( k-1 \) to the state at the current, \( k \)
- **\( H \)**: Output matrix relates the state to the measurement
- **\( \hat{x}_k \)**: Output of updated state
- **\( P_k \)**: Error covariance matrix of updated state
- **\( z_k \)**: Measurement
- **\( v_k \)**: Measurement noise
- **\( v \)**: Measurement noise covariance matrix (measurement error)
- **\( \hat{x}_k^- \)**: Initial state covariance
- **\( P_k^- \)**: Initial error covariance matrix

**From sensor spec or experiment**
Kalman Filter Made Easy by **MATLAB CVST**

**Initial state & covariance**

\[ \hat{x}_0, P_0 \]

**Previous state & covariance**

\[ \hat{x}_{k-1}, P_{k-1} \]

\[ k \rightarrow k-1 \]

Current becomes previous

---

**Time Update (“Predict”)**

\[ [z_{\text{pred}}, x_{\text{pred}}, P_{\text{pred}}] = \text{predict}(obj) \]

- \( z_{\text{pred}} \): prediction of measurement
- \( x_{\text{pred}} \): prediction of state
- \( P_{\text{pred}} \): state estimation error covariance at the next time step

---

**Measurement Update (“Correct”)**

\[ [z_{\text{corr}}, x_{\text{corr}}, P_{\text{corr}}] = \text{correct}(obj, z) \]

- \( z_{\text{corr}} \): correction of measurement
- \( x_{\text{corr}} \): correction of state
- \( P_{\text{corr}} \): state estimation error covariance

---

**Predicated state**

\[ x_{\text{pred}} \]

**Output of updated state**

\[ x_{\text{corr}}, P_{\text{corr}} \]

**Current Measurement**

\[ z \]

---

**Code Example**

```matlab
```
Sensor fusion algorithm for FCW

Data Pre-processing
- Calculate Ground Speed
- Object classification

Path Estimation
- Zoning

Sensor Fusion & Tracking
- Sensor Fusion
- Kalman Filter

Threat Assessment

Sensor types:
- Radar Object
- Vision Object
- Vision LD
- Vehicle CAN

Operations:
- Filtering
- Offset Compensation

Kalman Filter

Path Estimation
Reduce time on the road by *synthesizing* data to test algorithms

- **Synthetic data**
  - Logged vehicle data
  - ADAS algorithm
  - Expected behavior?
    - yes
    - no
  - Refine algorithm
  - Drive and collect more vehicle data
  - Create new traffic scenario or refine sensor model

- Generate code
- C Code
- Integrate with embedded environment
Common approaches to synthesizing test data

“Buy It”
- Buying “Off the shelf” solutions like PreScan and CarMaker enable you to author traffic scenarios and synthesize sensor data.

“Build It”
- Building it yourself enables you to control the level of fidelity/complexity appropriate for your application.
ADAS System Level Simulation – Lane Keeping Support
Model-Based Design for ADAS & Automated Driving

- Vision Algorithm
- Control Algorithm
- Sensor Data Algorithm
- Camera Sensor
- Traffic Infrastructure
- Vehicle Dynamics
- Radar Sensor
- Radar Algorithm
- Sensor Fusion & Situational Analysis
- Control Algorithm
- Camera Sensor
- Radar Sensor
- Control Algorithm
- Sensor Data Algorithm
- Vision Algorithm
Example synthetic test scenarios – intersection and stop & go

- Intersection – turning w/ braking
- Intersection – near crash
- Intersection – U-turn
- Stop & go @ 30→20kph

50kph

Ego car
Target car
Key components defining a Traffic Scenario

**Define Road**
- Road type
- NumLanes
- LaneWidth

---

**Define Vehicles**
- Ego vehicle
- Target vehicles
- NumTargets

---

**Define Trajectories**
- Path type
- Waypoints
- Velocity, acceleration
- Heading angle, yawrate

---

**Sensor Model**
- Vision, Radar
- Range, FoV
- Position, Velocity error
- NumCluster
Example radar spec

<table>
<thead>
<tr>
<th>Parameter</th>
<th>ESR Long-Range Requirement</th>
<th>ESR Mid-Range Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Range</td>
<td>&lt; 1m (&gt; 10dB target)</td>
<td>&lt; 1m (&gt; 10dB target)</td>
</tr>
<tr>
<td></td>
<td>&lt; 1m (&gt; 0dB target)</td>
<td>&lt; 1m (&gt; 0dB target)</td>
</tr>
<tr>
<td>Maximum Range</td>
<td>&gt; 175m (&gt; 10dB target)</td>
<td>&gt; 60m (&gt; 10dB target)</td>
</tr>
<tr>
<td></td>
<td>&gt; 100m (&gt; 0dB target)</td>
<td>&gt; 50m (&gt; 0dB target)</td>
</tr>
<tr>
<td>Range Accuracy</td>
<td>&lt; +/-0.5m noise component with +/- 5% bias component</td>
<td>&lt; +/-0.25m noise component with +/- 5% bias component</td>
</tr>
<tr>
<td>Range Discrimination for two targets at same angle &amp; range rate</td>
<td>&lt; 2.5m</td>
<td>&lt; 1.25m</td>
</tr>
<tr>
<td>Minimum Range Rate</td>
<td>&lt; -100m/s</td>
<td></td>
</tr>
<tr>
<td>Maximum Range Rate</td>
<td>&gt; +40m/s</td>
<td></td>
</tr>
<tr>
<td>Range Rate Accuracy</td>
<td>&lt; +/- 0.12m/s</td>
<td></td>
</tr>
</tbody>
</table>

Agenda

Workflow for ADAS algorithm development
  - Forward Collision Warning example
  - Sensor Fusion techniques

Sensor components modelling and simulation
  - Radars
  - Lidars
  - Cameras

Ground Truth Labeling
Integration in Model Based Design workflow
FMCW Radar: How Does it Work?

- Frequency modulated source (chirp)
  - Saw tooth, triangular, custom sweep
- Direct RF conversion (homodyne)
- MIMO: multiple antennas
What Behavior Can Be Modeled?

- Channel modeling (interference, noise)
- Modeling of antenna arrays (no of elements, position)
- Modeling of RF Impairments (noise, non-linearity, frequency dependency)
- Determination of waveforms (range, resolution)
- Algorithms for data analysis
Which MathWorks Tools Can Help?

- Phased Array System Toolbox
- Signal Processing Toolbox
- Instrument Control Toolbox
- Communications System Toolbox
- Antenna Toolbox
- LNA
- SimRF
- DSP System Toolbox
What is LiDAR

- **LiDAR**: **Light Detection And Ranging**
- Measures distance by illuminating environment
- Provides highly accurate distance measurements

Image courtesy Wikipedia (link)
Why use LiDAR for Autonomous Driving

Account for limitations of vision and radar sensors

Cameras perform poorly in bad weather or limited visibility.

Radar not efficient at detecting object classes.
Why use LiDAR for Autonomous Driving

LiDAR provides accurate distances and enables terrain and surface mapping

- Effective for autonomous path planning
Key Takeaways

1. Single environment for:
   - Data visualization and analysis
   - LiDAR algorithm development
   - Test and verification

2. Easiest way to get started with LiDAR processing

3. Integrate with autonomous robotic workflows
Developing Active Safety Systems

Developing the Vision Algorithm Part

- MATLAB based workflow allows
  - Easy to debug scripting language
  - Pixel level, 2D, 3D visualization
  - Easy-to-use powerful image processing, and computer vision algor...
Agenda

Workflow for ADAS algorithm development
  - Forward Collision Warning example
  - Sensor Fusion techniques

Sensor components modelling and simulation
  - Radars
  - Lidars
  - Cameras

Ground Truth Labeling
Integration in Model Based Design workflow
Stream Processing in MATLAB

System Objects:

• make stream processing easier and more efficient

• can accelerate your MATLAB code

• are very easy to create, configure and execute

• support floating and fixed-point data types as well as automatic C/C++ and HDL code generation

• Can be easily converted into Simulink blocks
Working with System Objects

```
>> Audiofilereader = dsp.AudioFileReader

Audiofilereader =

    System: dsp.AudioFileReader

Properties:
    Filename: 'C:\MATLAB\2015a\toolbox\dsp\dsp\speech_dft.mp3'
    PlayCount: 1
    SampleRate: 22050
    SamplesPerFrame: 1024
    OutputDataType: 'double'
```
How to create test bench in MATLAB

%% Create and Initialize
SamplesPerFrame = 1024;
Fs = 44100;

Microphone = dsp.AudioRecorder('SamplesPerFrame', SamplesPerFrame);
Spectra = dsp.SpectrumAnalyzer('SampleRate', Fs);

%% Stream processing loop
 tic;
 while toc < 20
   % Read frame from microphone
   audioIn = step(Microphone);

   % View audio spectrum
   step(Spectra, audioIn);
 End

%% Terminate
release(Microphone)
release(Spectra)
Batch processing

All the data

Work on all the data at once…

Deliver all at once
Systems Objects enable stream processing

- Each frame available as soon as processed
  - Reduced memory footprint
  - Near real-time / Life analysis

- Typical applications
  - Communications Systems
  - Audio / video signal processing
  - Data acquisition
Stream processing in MATLAB

- *Streaming techniques* process continuous data from a captured signal or large file by dividing it into “frames” and fully processes each frame before the next one arrives
  - Memory efficient

- Streaming algorithms in DSP System Toolbox provide
  - Implicit data buffering, state management and indexing
  - Simulation speed-up by reducing overhead
Where to find System Objects?

- DSP System Toolbox
- Computer Vision System Toolbox
- Phased Array System Toolbox
- Communications System Toolbox
Signal processing example for algorithm’s - performance comparison
Compare algorithm execution time with different implementations:

<table>
<thead>
<tr>
<th>Versions of the Algorithm</th>
<th>Elapsed Time (sec)</th>
<th>Acceleration Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Traditional MATLAB functions</td>
<td>18.0953</td>
<td>1.0000</td>
</tr>
<tr>
<td>2. System Objects in-lieu of MATLAB functions</td>
<td>3.6914</td>
<td>4.9020</td>
</tr>
<tr>
<td>3. MATLAB Coder MEX version</td>
<td>1.5241</td>
<td>11.8725</td>
</tr>
</tbody>
</table>
Accelerating algorithm execution

User’s Code

for k=1:max
x = fft(data)
y = 20*log1

Pre-allocation and vectorization

Pre-defined efficient implementations of algorithms

Generate MEX files automatically with MATLAB Coder

Parallel computations on multicore computers, GPUs, and clusters
Options to speeding up MATLAB algorithms:

- Toolbox Functions
- Toolbox Functions + PCT (parfor)
- System Objects
- System Objects + PCT (parfor)
- System Objects + Code Generation
- System Objects + Code Generation + PCT (parfor)

PCT: Parallel Computing Toolbox
Dual core machine used for this example

Faster
Create components in MATLAB and reuse them in Simulink

```matlab
[mostImportantObject, ...
 egoPath, ...
 radarObjects, ...
 visionObjects, ...
 fusedObjects, ...
 assessedThreats] = forwardCollisionWarning( ...
 lane, inertialMeasurementUnit, ...
 vision, radar, params, reset);
```
Enable coverage of MATLAB and C code with Simulink Verification and Validation

Collects coverage on MATLAB code in “Normal” mode

Collects coverage on generated C code in “Software in the loop” mode

```matlab
65 if(params.KF1.enable) % enable KalmanFilter for visionData?
66 ba = trackvisiondata by Kalman Filter
67 visionObjects = trackvisionobjects(visionSensor, params.KF1, reset);
68 else
69    visionObjects = visionSensor;
70 end
```

```
#65: if(params.KF1.enable) % enable KalmanFilter for visionData?

Decisions analyzed:

<table>
<thead>
<tr>
<th>Decision</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>83/3</td>
</tr>
<tr>
<td>true</td>
<td>0/83</td>
</tr>
</tbody>
</table>
```

```
8590 /* 'forwardCollisionWarning':68' if(params.KF1.enable) */
8591 if (params.KF1.enable)

Decisions analyzed:

<table>
<thead>
<tr>
<th>Decision</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>83/3</td>
</tr>
<tr>
<td>true</td>
<td>0/83</td>
</tr>
</tbody>
</table>
```

```
861 /* 'forwardCollisionWarning':69' visionObjects = visionSensor; */
```
Automate regression testing with Simulink Test

Specify tests and interact with results in the Test Manager.

Generate a report to share results.
Prototype on hardware with Simulink Real-Time
as seen in today’s Test drive your ADAS algorithms presentation

Algorithm Models

- Forward Collision Warning
- Autonomous Emergency Braking

Vehicle and Environment Models

11100101
Agenda

Workflow for ADAS algorithm development
  - Forward Collision Warning example
  - Sensor Fusion techniques

Sensor components modelling and simulation
  - Radars
  - Lidars
  - Cameras

Ground Truth Labeling
Integration in Model Based Design workflow
The MathWorks environment helps ADAS engineers …

- Gain insight by replaying and visualizing logged and synthetic vehicle data
- Efficiently model components and develop processing algorithms
- Speed up prototyping of algorithms by generating code
Developing Reliable ADAS and Autonomous Driving Functions in MATLAB and Simulink

Marco Roggero
Senior Application Engineer
marco.roggero@mathworks.de